

VIDEO SURVEILLANCE EN LIGNE

PROJET réalisé par

HUET Geoffroy
JOUBERT Adrien
POTHIN Jean Philippe

PROJET encadré par

LHOMMEAU Mehdi

Projet EI4, Avril 2013, Angers

REMERCIEMENTS

Nous tenons tout d'abord à remercier M. Mehdi LHOMMEAU, notre tuteur de Projet qui nous a accompagnés pendant toute la durée de notre travail. Grâce à ses conseils, aussi bien en matière de gestion de projet que sur les domaines techniques abordés, nous avons pu avancer sur le projet. Il a également su nous soutenir lors des différentes phases du projet.

PLAN

INTRODUCTION

Dans le cadre des Projets pédagogiques, nous avons choisi de nous intéresser à la vidéo surveillance en ligne.

La surveillance n'est plus une affaire d'inspection périodique des lieux, comme autrefois. Il existe maintenant les systèmes d'alarmes, de vidéosurveillance, qui aident les agents de surveillances, car ils peuvent désormais, en plus des rondes surveiller un maximum de zone depuis un simple écran de leur poste de sécurité.

L'homme seul étant quelque chose d'imparfait, une vidéo surveillance astucieuse l'aiderait dans ses procédés de contrôle.

Dans ce projet, nous nous sommes attardés sur la conception d'un système de vidéo surveillance en ligne, qui nécessite l'utilisation d'une webcam, et d'un navigateur web, et en particulier nous avons développé une partie du système avec la technologie NodeJS. Le caractère « en ligne » du système nécessitait aussi le développement d'un algorithme de traitement d'images qui puisse détecter les incohérences telles que les intrusions et d'en informer les clients concernés.

I. Présentation du projet

L'objectif de ce projet est de réaliser, à l'aide du framework JavaScript Node.js et de la technologie HTML 5, une application WEB permettant d'effectuer des opérations de vidéo surveillance à distance. L'application finale devra permettre à un utilisateur de s'enregistrer sur un site Web et d'utiliser la webcam de son ordinateur (ou la caméra d'un Smartphone) pour faire de la vidéo surveillance sans avoir à installer la moindre application. On pourra imaginer que l'application WEB envoie des alertes en cas de détection d'intrusions. De même, pour des raisons de sécurité les images pourront être sauvegardées sur un serveur distant (RapidShare, DropBox, …) ou sur la machine locale de surveillance.

1. Le produit et ses objectifs

Afin d'offrir un service d'application de vidéosurveillance en ligne efficace, le programme décrit ici offrira une fonction de télésurveillance supportant plusieurs client, offrant une identification et s'utilisant sans installation. Le programme sera ainsi accessible depuis le navigateur internet compatible avec la webcam (Chrome, Opera, etc).

2. Contexte global du projet

L'ISTIA, école d'Ingénieurs de l'Université d'Angers et centre de recherche de haut niveau, ne dispose pas actuellement de système de vidéo surveillance. Pourtant, elle possède du matériel coûteux dans ses locaux. Il est donc indispensable de prémunir les locaux de l'établissement.

Toutefois, le système de vidéo surveillance étudiée ici, peut aussi être très utile aux particuliers, car il leur suffit d'avoir une connexion internet, et une webcam pour préserver à moindres coûts leurs propriétés en leurs absences.

Il existe déjà plusieurs applications qui permette la surveillance en ligne, mais elles sont soit payant, soit il est nécessaire d'installer un logiciel.

3. Les intervenants

Maitrise d'ouvrage : Mehdi Lhommeau (Enseignant à l'ISTIA).

Maitrise d'œuvre : Geoffroy HUET, Adrien JOUBERT, Jean-Philippe POTHIN (Étudiants à l'ISTIA)

II. Identification et présentation du besoin

1. Environnement

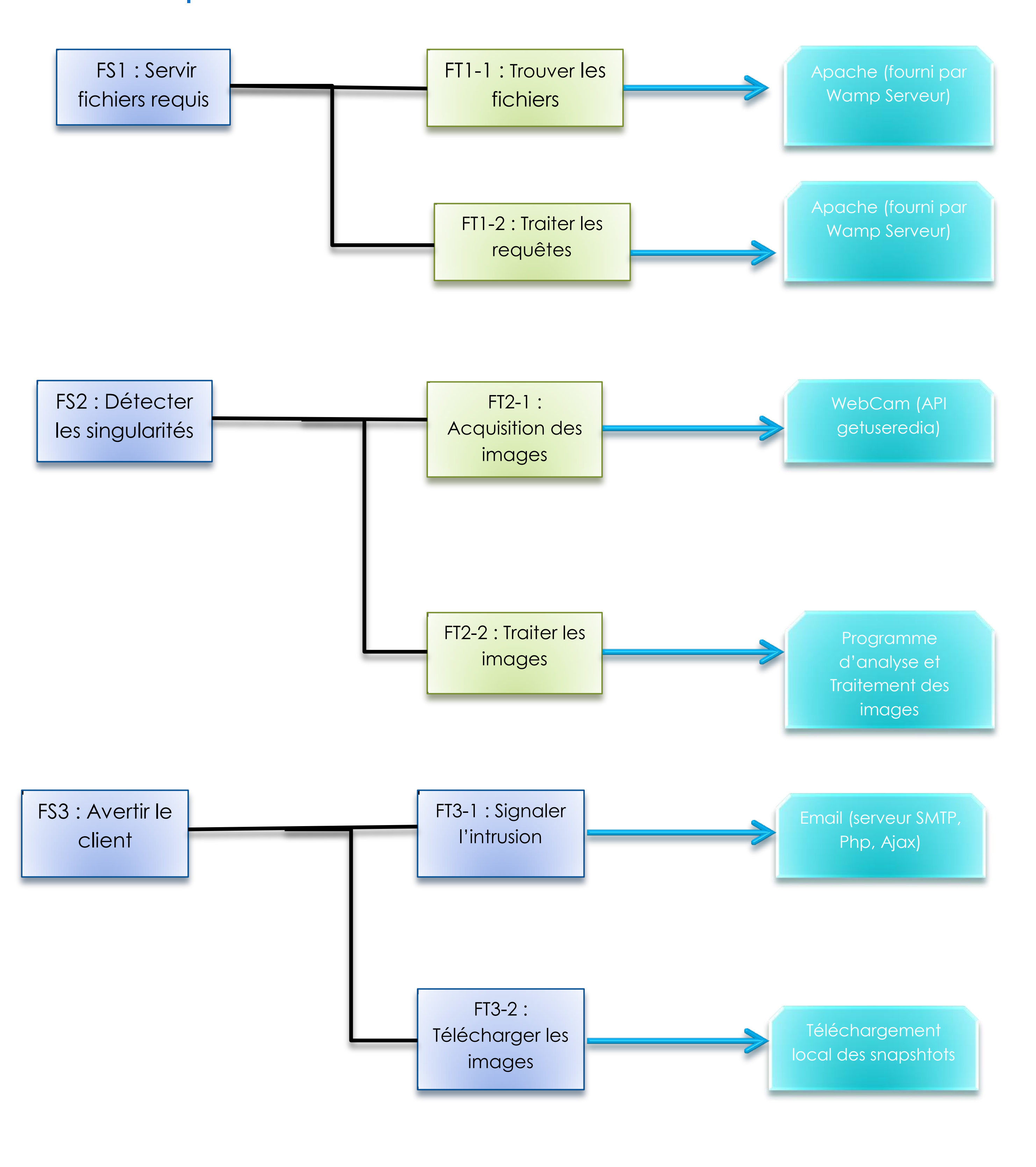
Ce projet concerne n'importe quel utilisateur possédant une connexion internet, une webcam ainsi qu'un navigateur compatible.

2. Analyse de la problématique

Tableau QQOQCP

Qui &/ou pour Qui ?	Tout public	• Premier système de vidéo surveillance • Coûts faibles
Quoi, avec Quoi ?	Vidéo surveillance en ligne	Webcam nécessaire et une connexion internet
Où ?	Propriétés des clients	
Comment ?	• Site web avec programme capable de détecter les singularités et d'avertir le client par email. • Possibilité de visualiser les images de la zone surveillée via Smartphone par streaming vidéo	

3. Description des fonctions



a) **Fonction Technique 2-1 : Acquisition des images :**

- On utilise les fonctions développées dans la dernière version de la suite HTML, soit la version 5, qui propose des fonctions de gestions des matériels d'acquisition vidéo (webcam) et audio (casque ou micro intégré).
- Prendre en compte la résolution, pour ne pas avoir une image trop floue et ne pas avoir de décalage de la vidéo lors du streaming.

b) **Fonction Technique 2-2 : Traiter les images :**

- L'analyse des images en cas d'intrusion doit être fait coter client, il est inutile d'encombrer le serveur avec les analyses, car le transit de flux vidéo encombre déjà assez le serveur.
- Lors du traitement, l'image doit être « binariser » car on soustrait deux images entre elles afin d'obtenir un seuil critique qui servira de référence au déclenchement des prises de photos lors d'une intrusion.

c) **Fonction Technique 3-1 : Signaler l'intrusion**

- En cas d'intrusion, une prise de photo doit s'exécuter automatiquement, suivit d'un envoi par email des photos prises.
- On doit pouvoir faire une distinction entre les adresses mails des clients.

d) **Fonction Technique 3-2 : Téléchargement local des photos**

De même que précédemment, mais l'intrusion déclenche en plus de l'envoi par emails, un téléchargement sur la machine locale des photos prises.

e) **Fonction Technique 4 (accessoire) : Accéder à la caméra depuis l'extérieur**

Depuis un autre ordinateur connecté à internet et muni de l'adresse email et du mot de passe décrit lors du lancement de l'application, il devra être possible de visionner en direct et à distance ce que filme la webcam.

f) **Fonction Technique 5 (accessoire) : Identifier et sécuriser**

- Pour lancer l'application : l'utilisateur doit communiquer son email. Il peut accessoirement indiquer un mot de passe s'il souhaite pouvoir visionner sa caméra depuis un autre ordinateur.
- Pour visionner une caméra sous surveillance : l'utilisateur devra indiquer l'email et le mot de passe qui auront été entré lors du lancement de l'application.

4. Description des contraintes

a) Contraintes organisationnelles

Dans une supposition de mise en ligne permanente, la mise en place du site devra alors nécessiter la location d'un serveur, d'un nom de domaine ainsi que d'un service d'emailing.

b) Contraintes techniques

Le serveur devra être sur liste blanche des adresses envoyant des emails afin que ses emails ne soient pas considérés comme du spam.

Le serveur devra disposer d'une bande passante lui permettant le multi-stream ainsi que l'emailing. Il devra également disposer d'un serveur SMTP.

L'utilisateur doit avoir un navigateur compatible pour la diffusion et la réception des flux.

Étant sur le web, l'application aura comme contraintes de résister aux formes d'intrusions les plus connus du grand public (injection SQL).

c) Contraintes réglementaires et légales

Comme toute application web, elle devra préserver la vie privée des utilisateurs et se plier à la loi relative à l'informatique, aux fichiers et aux libertés du 6 janvier 1978.

Le programme dont certaines parties sont sous licences LGPL ainsi que licence du MIT devra s'y conformer.

d) Contrainte des normes et standards

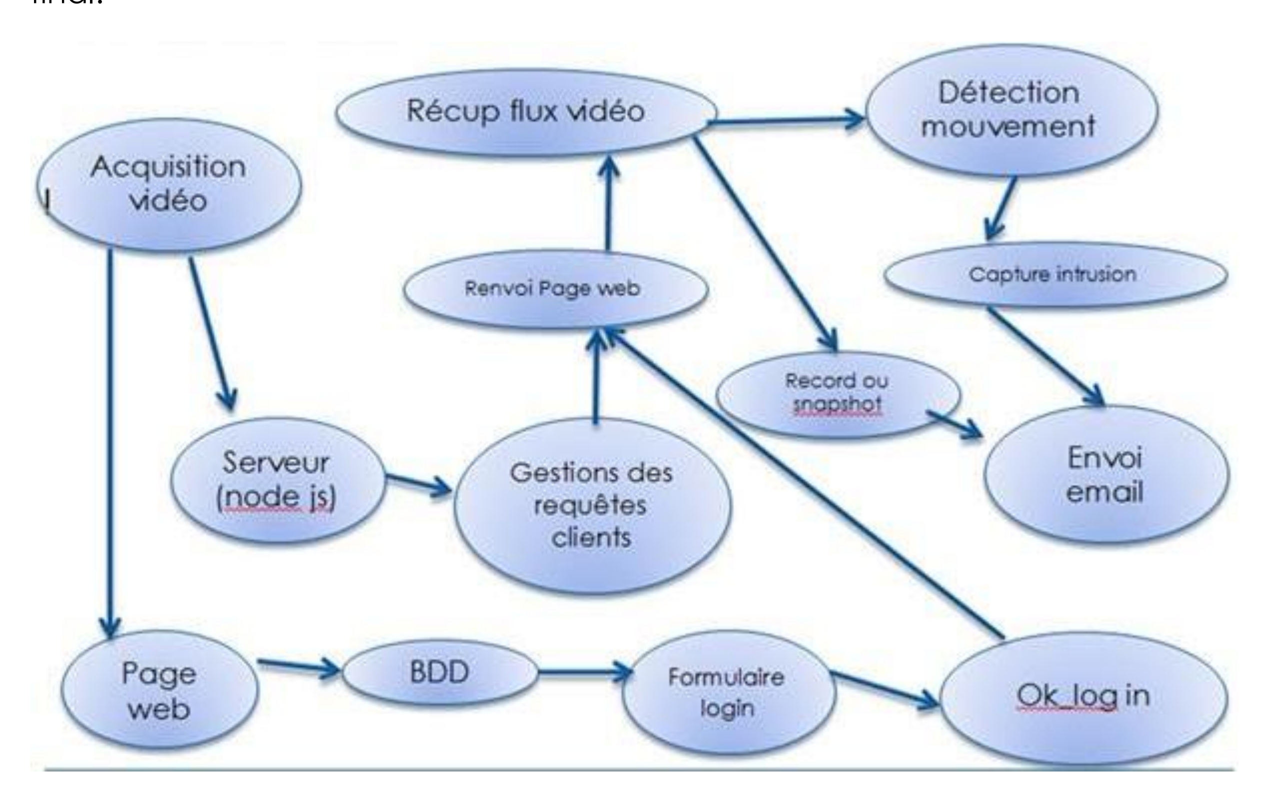
S'agissant d'un site internet, il devra s'appuyer sur les standards du web et devra se plier aux normes du W3C (World Wide Web Consortium).

III. Description de l'avant développement

1. Synthèse des outils de gestion

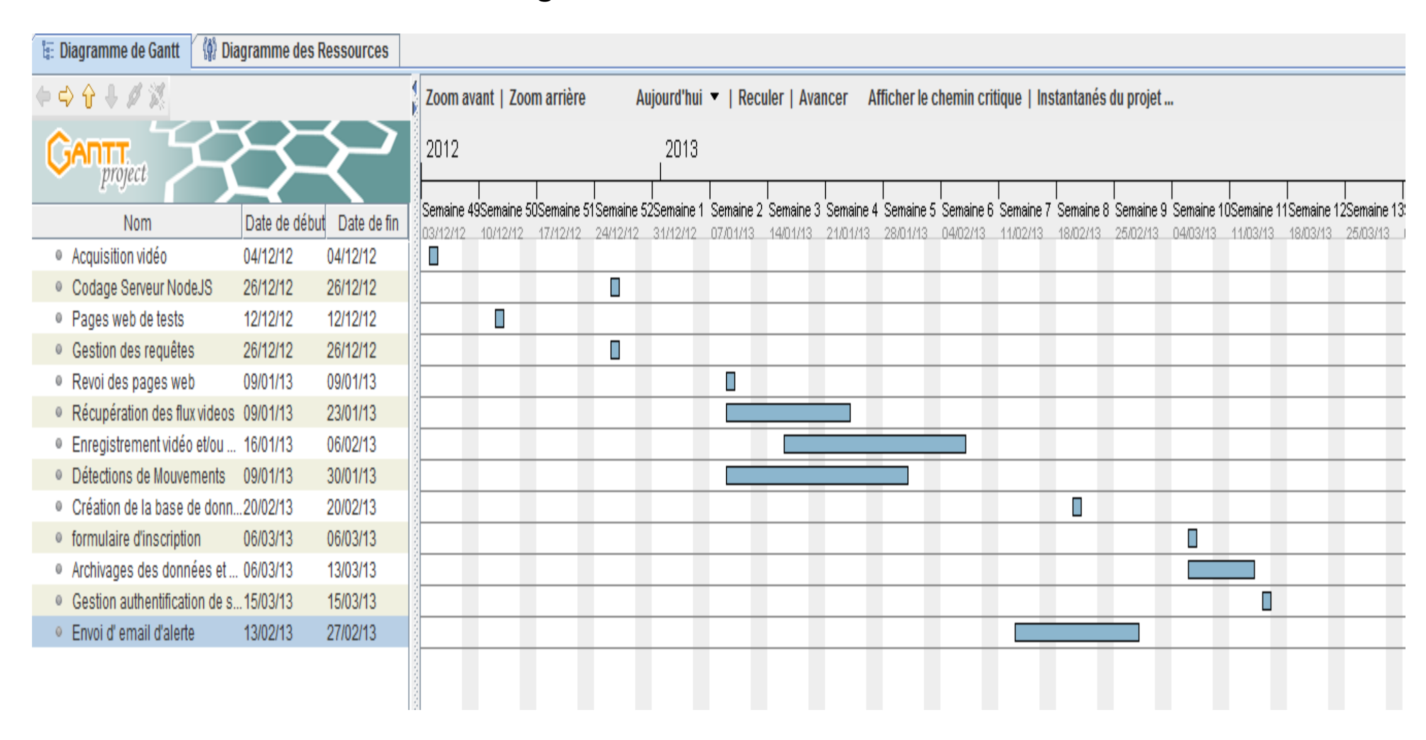
Grâce au diagramme de la partie II-2, on peut avoir une première vue (généralisée) du projet.

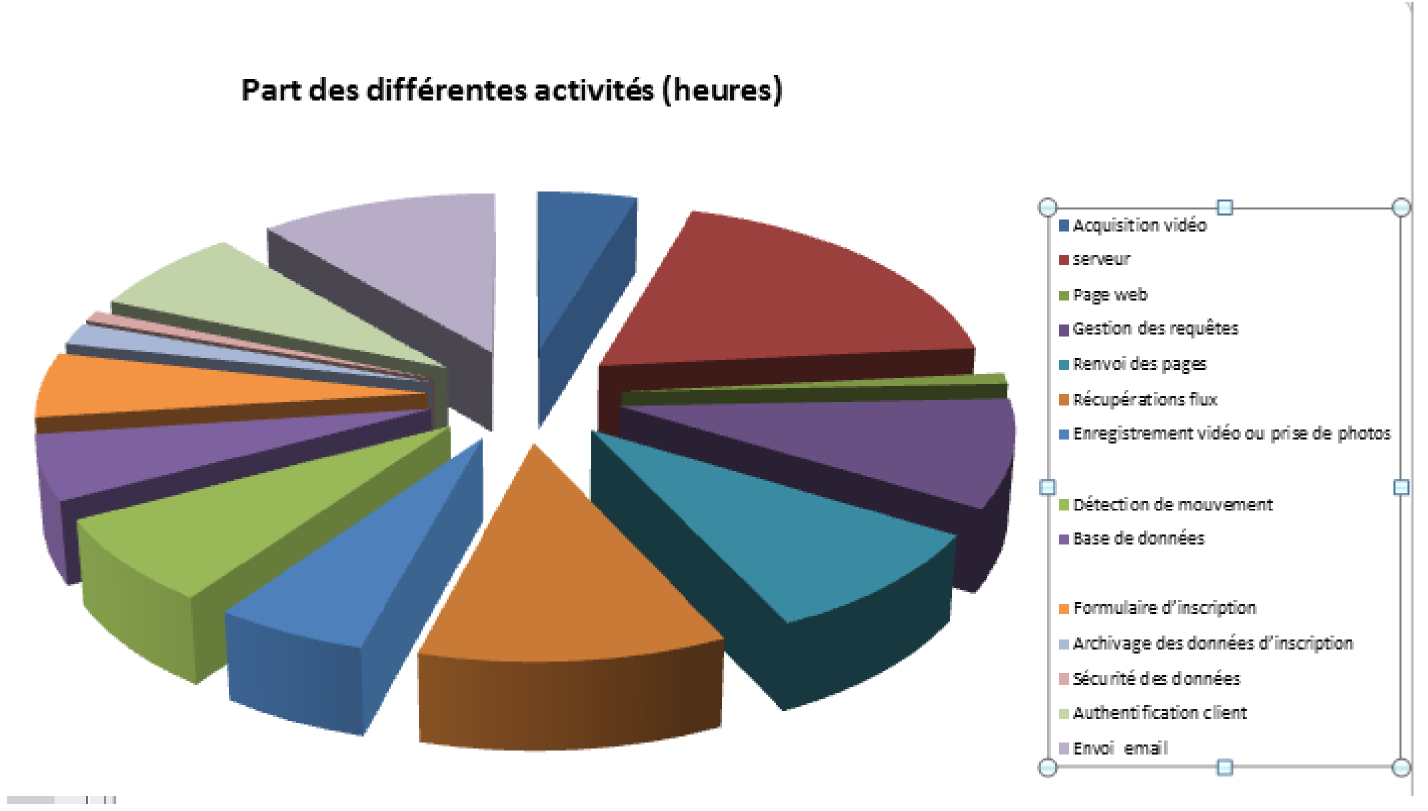
L'organigramme ci-dessous démontre le fonctionnement généralisé du système final.



Afin de mieux synthétiser le fonctionnement du système et mieux définir les fonctions constructives, du système de vidéosurveillance, on a représenté un diagramme FAST.

Diagramme de GANTT





2. Choix techniques

Nos choix technique ont été un compromis entre la consigne et la faisabilité/difficulté.

a) HTML5 oui, FLASH non

Le premier choix a été l'utilisation exclusive de l'HTML5 et aucune utilisation de Flash, propriétaire. Cela a augmenté la difficulté du projet dans pratiquement toutes ces phases. Cela entraîne une incompatibilité avec les anciens navigateurs (Internet explorer ainsi que les anciennes versions de Firefox) pour ce qui est de la détection de mouvement.

Cela a également forcé un choix autre technique ; celui concernant le streaming.

b) Utilisation de deux serveurs

Le second choix a été d'utiliser deux serveurs distincts, un serveur apache classique sous Windows (à travers Wamp) port 80 et un serveur de nouvelle génération nodeJS port 8080 permettant l'utilisation de JavaScript coté serveur. Ce nouveau type de serveur, novateur et très prometteur, permet de mieux supporter les demandes gourmandes.

Notre idée a été de gérer toutes les fonctions que nous considérons comme primordiale (identification, détection d'intrusion et envoi d'email) à l'aide du serveur Apache, utilisé depuis des années et réputé comme viable et d'utiliser le serveur Node pour le streaming car celui-ci dispose de modules spécifiquement développer pour permettant à travers des websocket le transfert facile de données.

c) Streaming centralisé et non P2P

Le P2P avec flash étant facile ; il est encore à l'état de brouillon sous Node. Il a été ainsi décidé d'utiliser un module à l'état de développement plus avancé et qui passe obligatoirement par le serveur (module ws pour websocket).

II-2-d) Choix bibliothèque nodeJS

Toutes les websockets proposés par la communauté se basent sur socketio. Chacune de ces implémentations rajoute une fonctionnalité spécifique à l'utilisateur et à son projet final. Notre choix s'est fait sur la version la plus authentique de socketio avec juste la possibilité d'envoyer des données et d'établir une connexion à défaut d'avoir trouvé une implémentation gérant à la volé la consommation BP du stream vidéo ainsi que la compression d'image.

d) Serveur SMTP local

Afin de se rendre indépendant des divers fournisseurs d'accès possibles de la futur plateforme du serveur ; nous avons choisis d'héberger en local notre propre serveur SMTP. Ce serveur permet l'envoi d'email direct. Ceci nous permet de gérer au mieux notre trafic.

e) Résolution maximum Webcam

Notre choix de résolution Webcam est le résultat d'un compromis entre la qualité d'image et la ressource nécessaire au calcul de mouvement. Ce choix est également pris selon la quantité de donnée à transmettre au serveur en cas d'intrusion.

f) Détection d'image en local

Afin d'alléger le serveur, toute la détection d'image est faite sur l'ordinateur du client (en JavaScript). Seul l'envoi de l'image par email est assuré par le serveur Apache.

3. Organisation du projet

A partir de l'analyse du besoin, de nos choix techniques et des outils de gestions, nous avons pu répartitif les taches et définir des règles d'organisation.

a) Découpage et planification des tâches

Pour la répartition des tâches, nous avons utilisés le planificateur d'équipe du logiciel Ms Project, dont voici l'affectation de chaque membre du projet aux taches:

HUET Geoffroy :

- Réalisation de la page web de test

- Création de la base de données

- Création du formulaire d'inscription et authentification des clients

- Archivages des données d'inscription et sécurité des données (intrusions)

- Elaboration du Design du site (feuille de style CSS)

JOUBERT Adrien :

- Acquisition des flux vidéo de la webcam

- Page web de test pour la visualisation des flux vidéo

- Détection des intrusions

- Paramétrage du serveur SMTP et envoi d'email d'alerte (Ajax)

- Envoi et Récupérations des flux vidéo pour le streaming (Nodejs)

POTHIN Jean Philippe :

- Codage du serveur écrit en NodeJS et gestion des requêtes clientes

- Pages web de test du serveur NodeJs

- Envoi et récupérations des flux vidéo pour le streaming

- Authentification des clients pour le streaming

- Gestion et organisation du projet

Node JS est un projet open-source qui est basé sur le moteur V8 de Chrome.

Il s'agit d'un interpréteur JavaScript, exécutable, et enrichissant le langage avec sa propre API

Son intérêt est d'interpréter du JavaScript coté serveur.

Contrairement au serveur Apache, où chaque requête HTTP entrante se voit allouer un thread qui est ensuite utilisé pendant toute la durée du traitement de la requête, Node n'utilise qu'un seul thread pour gérer les requêtes entrantes.

Ainsi, tout le code est asynchrone. L'avantage immédiat à cela est la simplification à l'extrême de la gestion de la concurrence dans nos applications qui n'est ici plus gérée par le développeur mais directement par l'OS.

Grâce à cela, on peut écrire des applications avec d'excellents temps de réponses.

b) Archivages des recherches

Les recherches sont partagés via un document sur Google drive. Elles sont classées par date ainsi que par source.

c) Sauvegardes des versions du projet

Deux répertoires existent :

- Un premier répertoire ou se trouve chacune des versions du programme. Une nouvelle version du programme est créée à chaque nouvelle fonctionnalité ajoutée.
- Un second qui regroupe chacune des fonctionnalités (prise à part du programme) ainsi que leurs évolution.

Cette architecture permet de créer et tester chaque fonctionnalité (comme l'envoi d'email ou la détection de mouvements) de façon séparé et isolé avant leurs rajouts au projet global.

d) Idées et solutions

Un autre document sur Google drive permet de stocker toutes les idées de chacun.

}

IV. Description du développement - phase de réalisation

1. Acquisition vidéo

L'acquisition des images de la webcam ne nécessite pas de logiciel spécial. En effet, depuis la sortie d'HTML5, cette nouvelle version permet de récupérer les flux vidéo et audio de tous les périphériques présent sur l'ordinateur à partir d'un simple navigateur web. Cependant, cette nouvelle fonction n'est pas totalement supportée par tous les navigateurs. Heureusement, trois des navigateurs les plus utilisés tels que Chrome, Opéra (dans le monde du mobile) et dernièrement Firefox supportent cette fonction.

navigator.getUserMedia (constraints, successCallback, errorCallback);

Cette fonction demande à l'utilisateur si il veut activer ses périphériques (webcam ou microphone).En cas de réponse positive, l'argument "successCallback" est alors invoqué.

La fonction getUserMedia appelle la fonction spécifiée dans le successCallback avec l'objet LocalMediaStream qui contient le flux multimédia. On peut assigner cet objet à l'élément approprié et travailler avec elle.

```
function setUpVideo() {
if (navigator.getUserMedia || navigator.webkitGetUserMedia) {
        if (navigator.getUserMedia) {
        navigator.getUserMedia('video',successCallback, errorCallback);
        function successCallback( stream ) {
        video.src=(navigator.getUserMedia?stream:window.webkitURL.createObjectURL(stream));
        ......
        }
        function errorCallback( error ) {    }
        } else {         }
}
```

Bien qu'optionnelle, la méthode errorCallback est nécessaire pour gérer une éventuelle absence de périphérique. Ce qui permettra d'informer l'utilisateur qu'il y a un problème.

Quant à la fonction sucessCallback, au moment où l'utilisateur autorise l'utilisation de sa webcam, on crée un objet URL dont la durée de vie est liée çà la fenêtre (window).

Et finalement pour le paramètre "vidéo" qui correspond à l'argument "constraint" de la fonction getUsermedia contraint juste à utiliser la fonction vidéo, soit la webcam.

La donnée des deux méthodes (callback) et l'utilisation de la contrainte "vidéo" nous permet de compléter la fonction SetupVideo qui permet de récupérer les vidéos de la webcam, qui sera appelé dès l'ouverture de la page web.

2. Détection de mouvements

La détection s'effectue en JavaScript. Voici le principe de calcul : pour détecter les images, on va soustraire pixel par pixel l'image actuelle avec l'image précédente. On va ensuite lire l'image finale. Si tous ses pixels sont noirs, il n'y aura pas eu de mouvement. Sinon, on verra des zones de lumières car un mouvement aura alors crée un décalage.

Voici la base du programme de détection de mouvement. Il est largement basé sur des programmes déjà existants sur le web. Voici sa description.

En entrée, il y a trois tableaux "Target", "data1"et data2 :

- Target contient le résultat de la soustraction

- data1 contient l'image actuelle de la webcam

- data2 contient l'image précédente de la webcam

Ces tableaux sont "aplatis" et contiennent les valeurs de l'image à la fois dans le rouge, le vert, le bleu et l'alpha. L'ordre est tel qui suit : pixels[0] = rouge, pixels[1]=vert, pixels[2]=bleu, pixel[3]=alpha, pixel[4]=rouge pixel n°2, pixel[5]=vert pixel n°2... ainsi de suite Il y a donc 4 cases par pixel.

Dans la fonction suivante "**differenceAccuracy**" , après moyenage des 3 éléments de chaque pixel, nous soustrairons les deux images en appliquant un filtre de min/max (avec un seuil de gris chois issu de tests) afin d'obtenir une image composé

uniquement de noir et de blanc. Le but est de calculer rapidement des différences issues de mouvements. Ces différences image d'un mouvement sont exprimés en pourcentage à travers la variable "**indicateurMouvements**". Le seul de détection à partir duquel un email sera envoyé est de 2%.

Voici le programme de détection de mouvement :

```
function differenceAccuracy (Target, data1, data2)

{

        if (data1.length != data2.length)

                return -1;

        var i = 0;

        var indicateurMouvements = 0;

        while (i < (data1.length * 0.25))

        {

                var average1 = (data1[4*i] + data1[4*i+1] + data1[4*i+2]) / 3;

                var average2 = (data2[4*i] + data2[4*i+1] + data2[4*i+2]) / 3;

                var diff = threshold(fastAbs(average1 - average2));

                Target[4*i] = diff;

                Target[4*i+1] = diff;

                Target[4*i+2] = diff;

                Target[4*i+3] = 0xFF;

                ++i;

                if (diff==0xFF)

                        indicateurMouvements++;

        }

        indicateurMouvements=(indicateurMouvements/data1.length)*100;

        return indicateurMouvements;

}
```

```
function threshold(value)
{
	return (value > 0x15) ? 0xFF : 0;
}
```

Le programme appelant celui-ci est exécuté 30 fois par seconde et envoi un email si le seuil de detection de 2% est dépassé.

3. Envoi d'emails

Lors de la détection d'une intrusion, l'image est communiquée au serveur en Ajax. Celui-ci, à l'aide de son serveur SMTP peut alors envoyer l'image en attaché à l'émail indiqué lors du lancement du stream.

Voici la fonction d'envoi de l'image en Ajax en JavaScript.

```
function envoyerMail()
{
	var canvasData = canvasSource.toDataURL("image/png");
	var ajax = new XMLHttpRequest();
	ajax.open("POST",'envoyerEmail.php',false);
	ajax.setRequestHeader('Content-Type', 'application/upload');
	ajax.send(canvasData);
}
```

Voici la fonction d'emailing en Php :

```
<?php
	// Reception de l'image
	$imageData=$GLOBALS['HTTP_RAW_POST_DATA'];
	// Supression des en-têtes
```

```
$filteredData=substr($imageData, strpos($imageData, ",")+1);

// On décode les données

$unencodedData=base64_decode($filteredData);

// On renseigne les coordonées de l'expédieur et de l'envoyeur

$email_expediteur='doNotReply@cctv.com';

$email_reply='doNotReply@cctv.com';

$message_texte='Bonjour,'."\n\n".'Un mouvement a été détecté. Vous trouverez ci-joint la photo associé.';

$destinataire='joubertadrien@gmail.com';//email de test (dans le futur, on ira chercher depuis le numéro de session php le véritable email dans la BDD)

$sujet="Surveillance WEBCAM : Mouvement détecté!";

$message_html='<html><head><title>Titre</title></head><body></body></html>';

//Frontiere entre email texte et html

$frontiere = '-----=' . md5(uniqid(mt_rand()));

//Headers de l'email

$headers = 'From: "Nom" <'.$email_expediteur.'>'."\n";

$headers .= 'Return-Path: <'.$email_reply.'>'."\n";

$headers .= 'MIME-Version: 1.0'."\n";

$headers .= 'Content-Type: multipart/mixed; boundary="'.$frontiere.'"';

//message coté texte

$message = 'This is a multi-part message in MIME format.'."\n\n";

$message .= '--'.$frontiere."\n";

$message .= 'Content-Type: text/plain; charset="iso-8859-1"'."\n";

$message .= 'Content-Transfer-Encoding: 8bit'."\n\n";

$message .= $message_texte."\n\n";

//Message coté html
```

```
	$message .= '--'.$frontiere."\n";
	$message .= 'Content-Type: text/html; charset="iso-8859-1"'."\n";
	$message .= 'Content-Transfer-Encoding: 8bit'."\n\n";
	$message .= $message_html."\n\n";
	$message .= '--'.$frontiere."\n";
	//On met l'image en piece jointe
	$message .= 'Content-Type: image/png; name="image.png"'."\n";
	$message .= 'Content-Transfer-Encoding: base64'."\n";
	$message .= 'Content-Disposition:attachement; filename="image.png"'."\n\n";
	$message .= chunk_split($filteredData)."\n";
	//Enfin on envoi l'email
	mail($destinataire,$sujet,$message,$headers);
?>
```

Le respect des headers et des adresses d'envoi/réception permet d'augmenter nos chances de ne pas être considéré en tant que SPAM par les grands domaines. L'IP de l'émetteur est également importante. Depuis l'ISTIA et avec ses implémentations nous pouvons ainsi envoyer avec aisance email vers Google ou Hotmail sans être considéré comme spam.

4. Base de donnée

La base de données MySQL est utilisée à l'aide de phpMyAdmin, fournis d'office avec Wamp. Voici la table users.

Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra
ID	int(11)			Non	*Aucun*	AUTO_INCREMENT
email	varchar(30)	latin1_swedish_ci		Non	*Aucun*	
mdp	varchar(30)	latin1_swedish_ci		Non	*Aucun*	
ID_session	varchar(30)	latin1_swedish_ci		Non	*Aucun*	
etat_stream	int(1)			Non	*Aucun*	

Nous trouvons dans la table users :

- Une clé primaire ID qui s'auto incrémente
- L'email de l'utilisateur de type varchar
- Le mdp crypté choisi par l'utilisateur de type varchar
- L'id de session de l'utilisateur de type varchar
- L'état du Stream, un booléen qui permet de savoir si l'utilisateur a choisi l'option Stream (nb : son utilité sera peut être modifiée d'ici la livraison)

5. Streaming

Le streaming, bien que fonctionnalité "bonus" est la partie la plus technique.

a) Principe

Notre système de vidéosurveillance doit permettre au client de visualiser la zone surveiller à n'importe quel moment, et n'importe où! Toutefois, cette possibilité n'est envisageable que si le client possède un Smartphone ou un ordinateur connecté à internet.

Pour pouvoir assurer le streaming des flux vidéo, on a utilisé un serveur codé en JavaScript utilisant l'API NodeJS.

Le principe du streaming est de capturer les vidéos depuis le navigateur et de faire transiter ce flux vidéo au server afin que celui-ci renvoi le flux vidéos à vers une page de réception.

Le client pourra alors accéder à cette vidéo via son Smartphone ou son ordinateur.

Cette brève description suggère deux choses.

- un canal de discussion entre la page d'envoi et le serveur
- un autre canal d'échange entre le serveur et la page de réception

Ainsi, le serveur ne stocke pas les flux vidéo, mais les redirige.

Pour pouvoir communiquer avec le serveur, on utilise une websocket. NodeJs propose plusieurs librairies de websocket toutes basées sur la librairie de départ à savoir socket.io.

b) Exemple d'utilisation

Voici ici un exemple simple du fonctionnement des websocket et de nodeJS.

Nous avons choisi d'utiliser websocket.io qui dépend bien évidement de socket.io, mais qui dispose de ses propres méthodes.

```
socket = new WebSocket('ws://localhost:8000');
socket.onopen = function() {
  setUpVideo();
};
```

La figure ci-dessus illustre l'ouverture du canal de communication depuis l'ordinateur qui envoi la vidéo vers le serveur.

"socket.onopen" est l'évènement qui est déclenché lors de l'ouverture de la socket. Il faut lui attacher un sous-programme. Par exemple, ici, setUpVideo() pourrait être un envoi en boucle de la vidéo. Ainsi dés lors de l'ouverture de la socket, la vidéo serait envoyé (socket.send(data)).

Evidemment, lors de l'hébergement du site, il faudra changer "localhost" par l'IP de la machine hébergeant le serveur nodeJS.

La vidéo est maintenant envoyée au serveur qui s'occupe de la rediriger vers le récepteur.

Ainsi, coté récepteur, on a le même système d'abonnement de code. Au lieu d'avoir un événement du type "socket.onopen", on a un événement "***socket.onmessage***" qui se déclenche lorsque le client reçoit des données de la part du serveur.

```
socket = new WebSocket('ws://localhost:8000');
socket.onmessage = function(event) {
  image.src = event.data;

};
```

On peut afficher les données de l'event contenant l'image directement dans un canva.

Examinons maintenant le code coté serveur: Ici se trouve un exemple d'une diffusion basique d'un streaming de vidéo à quiconque se connecte à la websocket.

```
72  var clients = [];
73  var websocketServer = ws.attach(httpServer);
74  websocketServer.on('connection', function (socket) {
75    clients.push(socket);
76    console.log('connected');
77    socket.on('message', function (data) {
78      clients.forEach(function(client) {
79        client.send(data);
80      });
81    });
82
83    socket.on('close', function () {
84      console.log('closed');
85    });
86
```

Ligne 73 . on associe notre variable webSocketServer au serveur http crée pour intercepter tous les requêtes clientes.

Ligne 74 : On associe à l'évènement 'connection' de la websocketServer nouvellement crée le sous-programme qui est décrit de la ligne 75 à la ligne 86+.

Ainsi dès qu'un nouveau client se connecte, il est enregistré dans le tableau "clients" et le message 'connected' s'affiche sur la console du serveur.

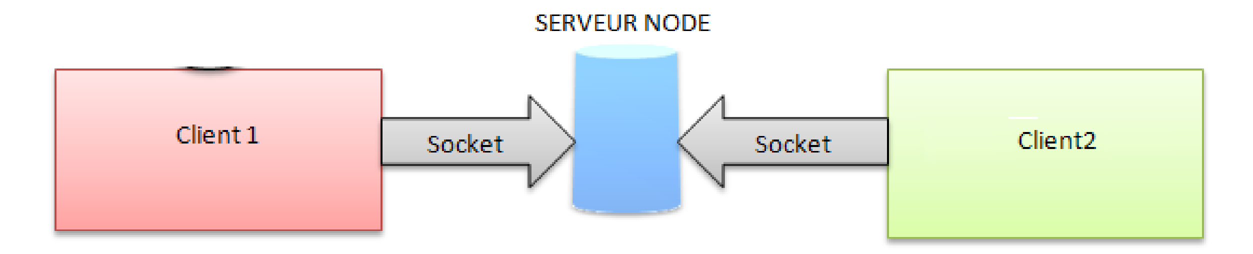
La ligne 77 à 81 décrit le programme qui doit s'exécuter dans le cas d'envoi de donné du client (note : dans notre programme final, on devra "selon qui il est" agir en conséquence). Ici qui que soit celui qui envoi les données, on les diffusera à tous ceux qui se sont un jour connecté sur le serveur.

c) Notre solution

Nb : notre solution à ce jour peut évoluer d'ici la présentation finale du projet.

La distinction des data envoyé par les clients a été la plus grande difficulté de ce projet. En effet, le cours de JavaScript ne faisant pas partie du cursus ISTIA, et l'API nodeJS étant encore nouvelle, nous avons passé du temps à appréhender le système d'abonnement de sous-programme et à, en finalité, discriminer les données envoyés par chaque client.

Pendant la phase de choix technologique, nous avions choisis de passer par le serveur. Voici la solution retenue :



Soit le client 1 étant un émetteur. C'est à dire un client qui utilise notre application pour détecter un intrus et étant ainsi à même d'envoyer des images.

Soit le client 2 étant un récepteur. C'est à dire un client qui utilise notre application pour réceptionner des images.

1) Le client 1 (si l'option "streaming" au sein de sa page a été activée) envoi trois arguments :

- Son type (0 s'il est un émetteur et 1 s'il est récepteur) sera 0 dans notre cas
- Son identifiant (un md5 du MDP et son email)
- La vidéo

2) Le serveur enregistre ce nouveau client et attends qu'un client du type récepteur arrive.

3) Le client 2 se connecte. Il déclare son type et son identifiant auprès du serveur.

4) Le serveur observe une correspondance entre client 1 et 2 et envoie la vidéo du 1 vers le client 2.

5) Le client 2 réceptionne les données (socket.on('message'... et les affiches

nb : dans le futur, il est prévu que le client 1 n'envoie des données que lorsqu'un client2 est abonné.

Le principe de codage de nodeJS et socketio fait que la partie 1 et 3 sont à coder en même temps dans la même fonction. Pour discriminer ce que chacun doit faire, il faudra toujours vérifier l'argument premier, le “type”.

Expliquons-le avec le code développé; coté serveur, nous avons :

```
var websocketServer = ws.attach(httpServer);

var emeteurs=[];
var recepteurs=[];

// A chaque arrivé d'un nouveau client
websocketServer.on('connection', function (socket)
{
```

Deux tableaux associatifs. Un pour les émeteurs et un pour les récépteurs.

```
socket.on('message', function (data)
{
  //On récupere l'identifiant de la socket
    var identifiant="";
    var i;
    //console.log("Data length :"+data.length);
    for (i=2;i<data.length;i++)
    {
      if (data[i]==",")
        break;
      else
        identifiant+=data[i];
    }
```

L'identifiant correspond au type de client (émeteur ou recepteur). Il sera soit 1 soit 0. Quand une connection (grâce à la création d'un événement “connection”) est établie, on récupère l'identifiant de la socket .

```
    if (data[0]==0)
    {
      //console.log("Un émeteur est connecté");
      //On enregistre l'émeteur s'il est nouveau
      if (emeteurs[identifiant]==undefined)
      {
        //emeteurs.push(identifiant);
        emeteurs[identifiant]= socket;
        //on sort si l'émeteur viens d'arriver
      }
      //On envoie au recepteur s'il existe
      else
      {
        if (recepteurs[identifiant]!=undefined)
        {
          console.log("Un récepteur est détecté");
          //On fabrique l'image
          var donnees = "";
          for (i=i+1;i<data.length;i++)
          {
            donnees+=data[i];
          }
          recepteurs[identifiant].send(donnees);
        }
      }
    }
```

Si un émetteur se connecte et qu'il est nouveau (donc non définit), on le déclare. S'il est connu, on regarde si un récepteur s'est abonné à lui et si oui, on lui transfère la vidéo.

```
    else
    {
      //On cherche l'identifiant correspondant au MDP et Email récup
      //BDD on chope SESSIONIDAPPACHE depuis MAIL + MDP
      //On enregistre le recepteur s'il est nouveau
      if (recepteurs[identifiant]==undefined)
      {
        //recepteurs.push(identifiant);
        recepteurs[identifiant]= socket;
      }
    }

    //SERA EXECUTE A LA FERMETURE DE CE SOCKET
    socket.on('close', function () {
      console.log('Someone is gone');
    });
```

Si un récepteur et non un émetteur s'est connecté, on l'enregistre.

Coté Client:

Page de réception:

```
10  window.onload = function() {
11    image = document.getElementById('image');
12
13    socket = new WebSocket('ws://localhost:8080');
14    // On s'identifie auprés du serveur
15    socket.onopen = function()
16    {
17      var MonTableau = new Array("1","MEC1");
18      socket.send(MonTableau);
19    };
20    // On receptionne l'image
21    socket.onmessage = function(event)
22    {
23      image.src = event.data;
24      //alert(event.data[9]);
25    };
26  }
```

La page de réception ne diffère de la première version que par l’utilisation d’un tableau de socket qui est à la ligne 17,qu’on envoi au serveur via la socket de connection.

Dans le code présenté ci-dessus, le tableau ligne 17 prend en paramètres deux champs “1” pour l’identifiant et “MEC1” pour son pseudonyme par exemple.

Evidemment dans le code final, il faudra modifier les paramètres du tableau avec les variables adéquates.

Page d'envoi:

```
211     function start()
212     {
213         $(canvasSource).show();
214         $(canvasBlended).show();
215         update();
216         //Lancement du stream
217         if (activerStream ==1)
218         {
219             socket = new WebSocket('ws://127.0.0.1:8080');
220             socket.onopen = function()
221                 {
222                     streamVideo();
223                 };
224                 /*
225             socket.onmessage = function(event)
226                 {
227                     serverDemandeStream = event.data;
228                 };*/
229         }
230     }
```

Cette fonction est appelée dès la connexion à la page web (index).Les lignes 213 et 214 font appelles au canvas, qui contiennent les vidéos. S'en suis une activation du streaming après vérification si le client veut effectuer un streaming. Coté graphique, ce choix sera assuré par un checkbox, et on récupère sa valeur.

Une fois cette vérification faite, on établit la communication avec le serveur grâce à une socket, et on envoi la vidéo grâce à la fonction *StreamVideo()*.

Zoom sur la fonction *StreamVideo()* :

```
203    function streamVideo()
204    {
205      var MonTableau = new Array("0","MEC1", canvasSource.toDataURL());
206      socket.send(MonTableau);
207      //socket.send(canvasSource.toDataURL());
208      timeOut2 = setTimeout(streamVideo, 10);
209    }
```

Remarquez la ligne 205 qui crée un tableau associatif contenant l'identifiant du client ("1") ,son pseudonyme ("MEC1") ,et la vidéo qu'il compte envoyer qui a été préalablement convertit en URL.(fonction toDataURL fournit par HTML).

C'est ce tableau qui diffère de la première version (ligne 207) ou on envoyait directement les flux vidéo sans distinction. Avec le tableau associatif, on sait maintenant qui envoi la vidéo, et par post-traitement(côté serveur),on sait pour qui elles sont destinées.

6. Authentification et sécurité

Pour le système d'authentification, l'utilisateur qui arrive sur la page d'accueil à la possibilité de soit se connecter si il possède déjà un compte, soit de s'inscrire.

Pour l'inscription il devra fournir comme informations son adresse email ainsi que son mot de passe (2 fois afin de vérifier qu'il ne se trompe pas) .

Afin de tester si l'adresse email est valide, on utilise une regex ainsi que la fonction php preg_match:

```
$Syntaxe='#^[\w.-]+@[\w.-]+\.[a-zA-Z]{2,6}$#';
if(preg_match($Syntaxe,$mail))  {//si le format de l'adresse mail est bon
```

On vérifie également en ajax s'il n'existe pas déjà un utilisateur avec la même adresse email dans la base de données. Si c'est le cas, l'utilisateur ne peut pas valider et est prévenue que l'email est déjà utilisé.

Si l'adresse est finalement valide, le mot de passe est crypté grâce à la fonction PHP md5, et l'utilisateur est ajouté à la table users.

```
//création du user
$mdp=md5($mdp); //on cripte le mdp
mysql_query("INSERT INTO users (email, mdp) VALUES ('$mail','$mdp')") or die('Erreur SQL !<br>'.$sql.'<br>'.mysql_error());
```

Pour la connexion il devra également fournir son adresse mail et son mot de passe.

De même, on vérifiera si l'adresse email est valide.

On fera ensuite une requête select pour savoir si l'utilisateur existe bien:

```
$sql = 'SELECT mdp FROM users where email="'.$mail.'"'; //on selectionne l'user
$req = mysql_query($sql) or die('Erreur SQL !<br>'.$sql.'<br>'.mysql_error());
```

Ensuite afin d'éviter les injections SQL, on vérifie que la requête ne renvoie qu'une seule réponse:

```
if(mysql_num_rows($req)==1){
```

Si c'est le cas on comparera le mot de passe fournie par l'utilisateur (qu'on aura préalablement crypté) et celui de la base de données. S'ils sont égaux on passera une variable de session à 1 et on redirigera l'utilisateur vers la page principale:

```
if ($data['mdp']==""){//si l'user n'existe pas
    $_SESSION['erreur'] = "Erreur email/mot de passe";
    header('Location: index.php');
}
else{//si l'user existe
    if ($data['mdp']==$mdp){//mdp bon
        $_SESSION['niveau']=1;
```

Cette page affichera son contenu uniquement si la variable de session est égale à 1, sinon elle proposera un lien vers la page d'authentification.

Pour le test de vérification afin que les champs soient bien remplis par l'utilisateur, on utilise l'attribut "required" fournis par HTML5.

```
<input style="height:40px;" type="text" placeholder="Email"
required="" name="username" id="username" />
```

Enfin, afin de ne pas se faire pirater la base de données, on créer un utilisateur qui a uniquement les droit de sélection, d'insertion et d'update.



user	%	Oui	SELECT, INSERT, UPDATE

7. Téléchargement de la photo

Lors de la détection de mouvement, on ouvre un pop-up qui téléchargera la photo grâce à la fonction window.open():

```
window.open ('popup.php', 'DL capture', config='height=500, width=500, toolbar=no,
menubar=no, scrollbars=no, resizable=no, location=no, directories=no, status=no')
```

Le fichier popup.php contient le code suivant qui permet de télécharger la photo directement sur l'ordinateur de l'utilisateur. Le nom de la photo sera l'email de l'utilisateur.:

```
$filename=$_SESSION['email'].".png";

// Envoi du fichier
header('Content-Transfer-Encoding: none');
header('Content-Type: application/octetstream; name="'.$filename.'"');
header('Content-Disposition: attachment; filename="'.$filename.'"');
header('Content-length: '.filesize($filename));
header("Pragma: no-cache");
header("Cache-Control: must-revalidate, post-check=0, pre-check=0, public");
header("Expires: 0");
@readfile($filename) OR die();
```

Enfin on ferme le pop-up après 5s (5s étant le temps jugé nécessaire au téléchargement de l'image) grâce au code JavaScript:

```
function closewindow() {
    self.close();
}

setTimeout(closewindow, 5000);
```

Le setTimeout permet d'appeler la fonction closewindow après un délais de 5 secondes.

V. Description de l'après développement:

1. Présentation du programme

Présentons les résultats sous chaque aspect du projet.



Tout individu voulant essayer la vidéo surveillance en ligne devra s'identifier ou tout simplement créer un compte en spécifiant son adresse email et son mot de passe.

Surveillance video en ligne

☐ Activer le streaming

Démarrer la surveillance

Déconnexion

Quitter la surveillance

Choisissez un délais avant le lancement de la surveillance (en secondes):

Timer

Démarrer

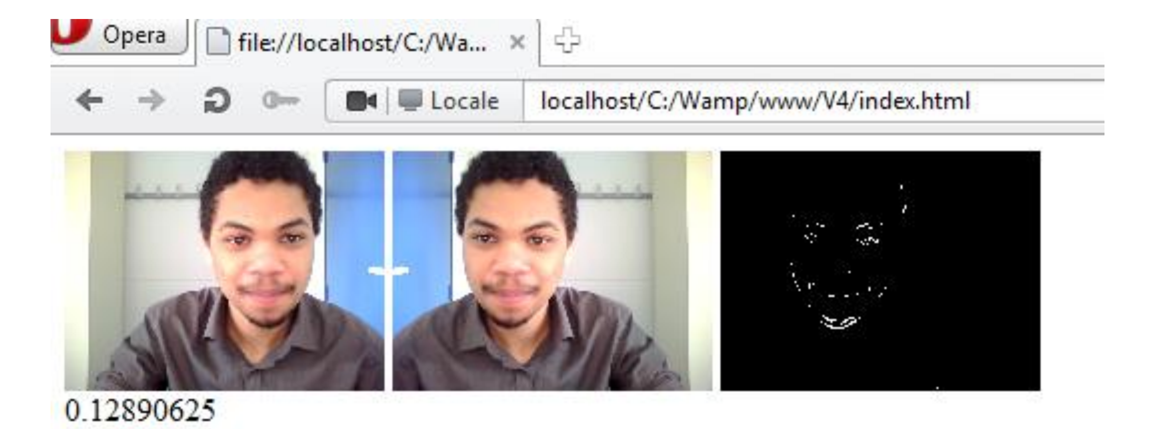
Voici un premier aspect du rendu de l'interface:

La checkbox pour activer le streaming, un bouton pour pouvoir démarrer la surveillance tout de suite, ou sinon le client à la possibilité de préciser un timer qui déclenchera la vidéosurveillance dès la fin du timer.

Et enfin, l'utilisateur peut à tout moment quitter, abandonner la surveillance et se déconnecter.

La fenêtre de surveillance (avec la vidéo) sera située juste en dessous du bouton démarrer.

Le design du site reste sobre pour l'instant, le rendu final (pour le jour de la soutenance) sera plus esthétique.



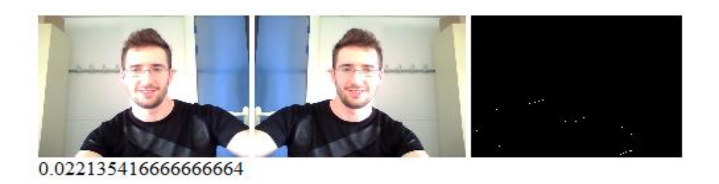
Sur cette copie d'écran, on peut remarquer le test de l'algorithme de traitement d'image.

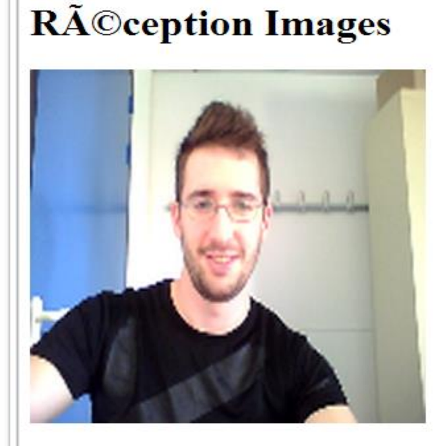
De gauche à droite, on peut voir la première image, puis une seconde image bien évidement différente. En Noir et blanc, l'image « binarisée », après soustraction des deux.

Le pourcentage de différence entre les deux images, ici n'est que que de 0.13%,ce qui ne déclenchera pas de prise de photo et d'envoi d'email d'alerte car le seuil de détection choisi est de 2%.



Par contre sur la capture ci-contre, le seuil est de 4.33723958333%, de différence entre les deux images, et ce pourcentage est au-dessus du seuil, donc un email d'alerte sera envoyé avec la photo en pièce jointe.





Sur cette copie, on peut voir un aperçu du streaming, avec à gauche la page d'envoi, et à droite, la page de réception. Attention tout de même au navigateur utilisé; le streaming ne marche qu'avec les navigateurs chrome et opéra.

2. Manuel d'utilisation (coté client)

Pour pouvoir utiliser cette application de vidéosurveillance en ligne, le client doit vérifier tout d'abord ses quelques points:

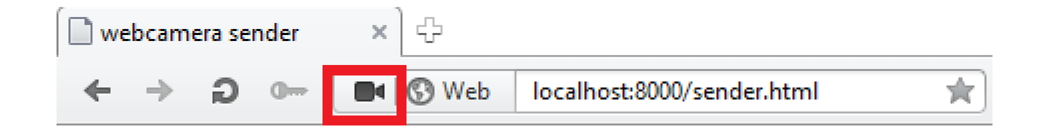
- avoir une webcam fonctionnelle (intégré ou non)

- avoir au moins le navigateur chrome ou opéra d'installer sur sa machine et son Smartphone.

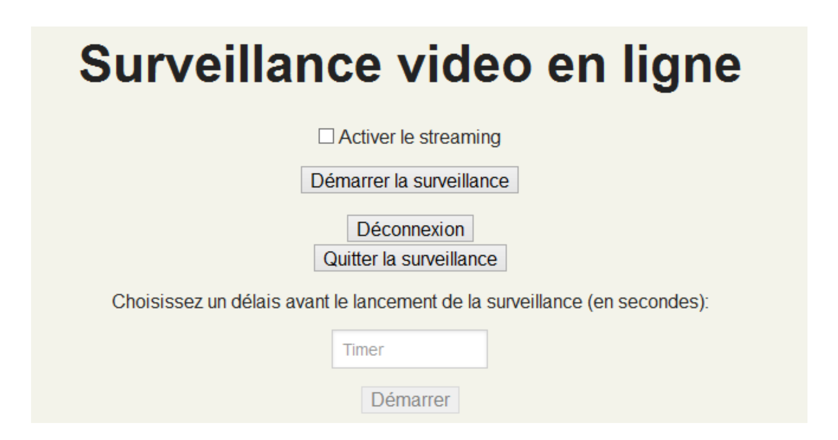
- Le client doit avoir une adresse email valide.

Avec ses points, l'utilisateur pourra se connecter sur le site homemonitoring.com (bientôt valide),il devra s'identifier, ou sinon créer un compte utilisateur.

Une fois connecté, le navigateur interrogera le client pour pouvoir lui demander l'utilisation de sa webcam, et il devra répondre positivement à cette demande.



Puis l'utilisateur sera redirigé sur la page de surveillance à proprement parlé.



Choisir les options désirées et démarrer la surveillance et n'oublier pas de vérifier vos emails de temps en temps.

3. Manuel d'installation (coté serveur)

La machine qui hébergera l'application doit avoir l'API nodeJS d'installé (http://nodejs.org/)

Il lui faudra également Wamp serveur avec la base déclaré sous phpmyadmin ainsi qu'un serveur SMTP pour l'envoi d'emails.

Il lui faudra enfin la copie de tous nos sources. Une fois Apache et Node lancé, le serveur est prêt !

CONCLUSION

1. Programme attendu/programme final

Rappel des objectifs:

L'objectif de ce projet est de réaliser, à l'aide du framework JavaScript Node.js et la technologie HTML 5, une application WEB permettant d'effectuer des opérations de vidéo surveillance à distance.

L'application finale devra permettre à un utilisateur de s'enregistrer sur un site Web et d'utiliser la webcam de son ordinateur (ou la caméra d'un Smartphone) pour faire de la vidéo surveillance sans avoir à installer la moindre application.

On pourra imaginer que l'application WEB envoie des alertes en cas de détection d'intrusion. De même, pour des raisons de sécurité les images pourront être sauvegardées sur un serveur distant (RapidShare, DropBox, …).

De même, dans la version finale, on pourra tester l'application sur une carte embarquée de type Raspberry PI.

Par rapport aux objectifs attendu, le rendu final du projet réponds aux exigences suivantes:

- opérations de vidéosurveillance
- Enregistrement sur le site web
- Utilisation de la webcam pour des opérations de vidéosurveillance
- Aucune application à installer
- Envoi d'alerte d'intrusion (email)
- Téléchargement des photos d'intrusions sur la machine locale ainsi que des vidéos

Par contre, pour des raisons de confidentialités et de sécurité, et pour respecter la loi concernant la vie privée, nous n'avons pas voulus sauvegarder les images d'intrusions sur un serveur distant tel que DropBox ou RapidShare. En effet, nous avons préféré envoyer les images en pièce jointes dans les emails d'alertes.

Un des bémols de ce projet était la visualisation des flux vidéo, donc du streaming sur un Smartphone. Cette opération ne fonctionne malheureusement pas totalement, car il faut que les Smartphones aient tous le navigateur opéra d'installer, ou à défaut le navigateur chrome.

Comme précisé précédemment, d'un point de vue technologique, nous avons réalisé ce projet à l'aide du framework JavaScript Node.js pour la création du serveur et de la technologie HTML 5 pour ce qui est de l'utilisation de la webcam.

Enfin nous n'avons pas eu le temps de mettre en place le teste de l'application sur une carte embarquée de type Raspberry PI.

2. Besoins non retenus

La communication en email d'une vidéo (en plus de la photo).

3. Améliorations

Une plus large gestion des streaming. Le lot 1 pourra traiter de la compression de vidéo à la volée ainsi que la communication P2P entre 2 client (déchargeant le serveur tout en améliorants la connexion comme la qualité de la vidéo). La stabilité du serveur pourra également être améliorée.

BILAN DU PROJET

Ce projet nous a permis de programmer en JavaScript, un cours qui ne fait pas partie du programme de programmation internet de l'ISTIA. Les connaissances en programmation web utilisées ont été multiples et ce projet a été l'occasion d'en faire la synthèse et de les appliquées. Cela nous a permis d'enrichir notre expérience.

De même, on a pu découvrir que faire du JavaScript coté serveur est désormais possible grâce à l'API NodeJS.

Le travail en équipe étant quelque chose d'essentiel pour un ingénieur, il était aussi très intéressant de confronter les idées sur une situation qui bloquait l'avancement du projet, et en particulier sur le streaming.

Pour finir, cette expérience nous a permis de développer notre capacité d'adaptation, dans un environnement où nous n'avions pas particulièrement de connaissances ce qui pourra nous servir d'expérience lors d'insertion dans le monde professionnel et notamment lors de nos premières périodes de stage.

Webographie

http://nodejs.org/

http://my.opera.com/core/blog/2011/03/23/webcam-orientation-preview

http://people.opera.com/danield/html5/explode/

http://www.quietless.com/kitchen/building-a-login-system-in-node-js-and-mongodb/

http://www.adrienbaptiste.com/developpement/creer-un-chat-facebook-avec-nodejs-nowjs/671

http://www.adrienbaptiste.com/developpement/creer-un-chat-nodejs-avec-login-mysql/920

http://www.emergence-group.com/en/node/426

http://www.adrienbaptiste.com/developpement/creer-un-chat-facebook-avec-nodejs-nowjs/671

http://www.grafikart.fr/tutoriels/nodejs/nodejs-socketio-tchat-366

https://github.com/search?q=webcam+nodejs&type=&ref=simplesearch

http://naholyr.fr/2011/07/authentification-et-websocket-avec-node-js-express-et-socket-io/http://w

http://ericbidelman.tumblr.com/post/31486670538/creating-webm-video-from-getusermedia

http://stackoverflow.com/search?q=nodejs

Résumé

Notre projet consiste à créer une application web qui fournit la surveillance d'une pièce grâce à l'utilisation de la webcam. L'utilisateur doit seulement se connecter suite à une inscription, et ensuite lancer la surveillance. Lors du lancement il a le choix d'activer le streaming, ce qui lui permet via une autre machine d'avoir accès aux images de sa webcam. Lors de la détection de l'intrusion, un email est envoyé à l'utilisateur avec l'image capturée. Les langages utilisés pour cette application sont le JavaScript, particulièrement pour coder le serveur, ainsi que le php et l'HTML. Pour le codage du serveur nous avons utilisé le framework NodeJS.

Mot clés : NodeJS, JavaScript, HTML5, Surveillance

Summary

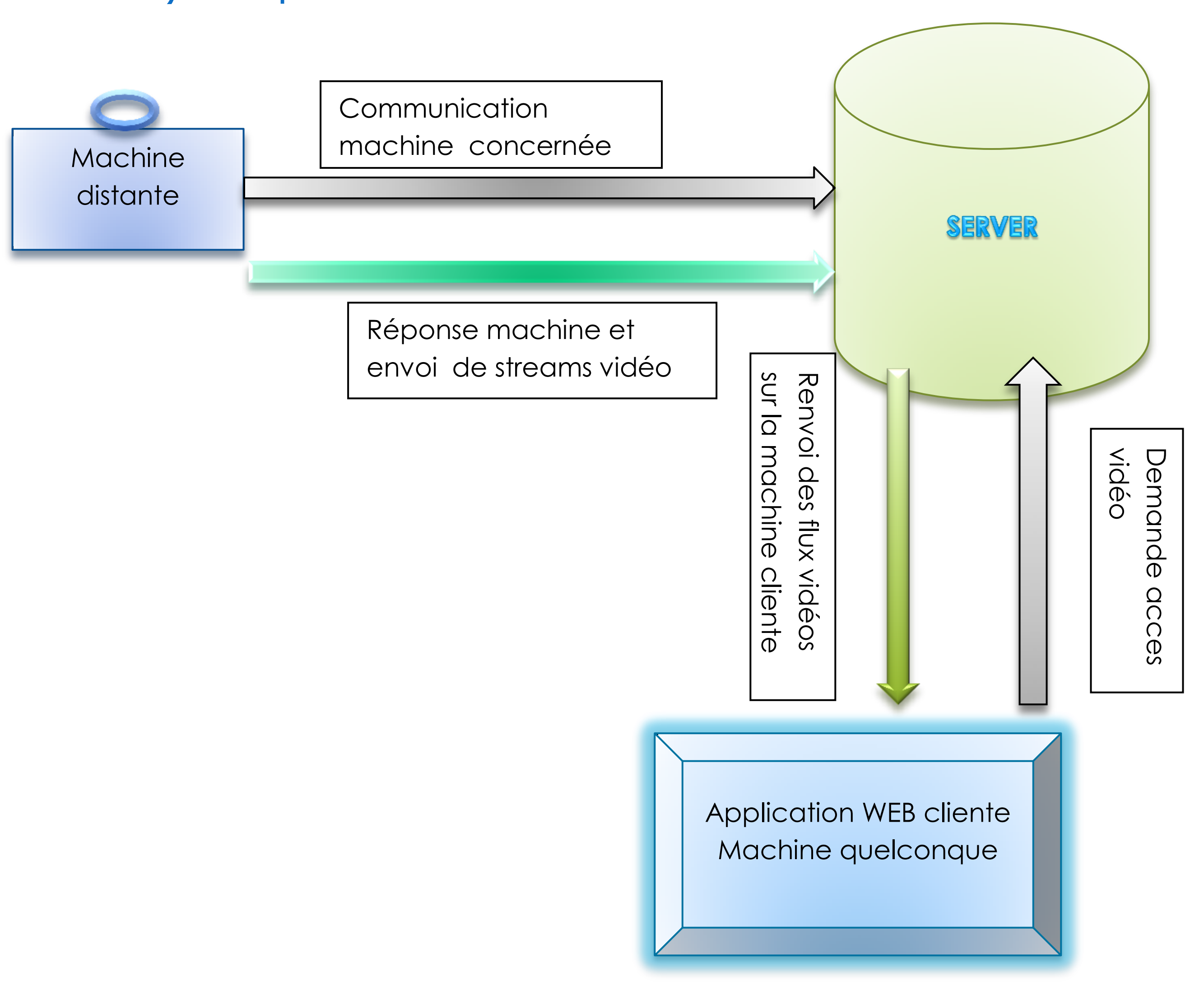
Our project is to create a web application that provides monitoring of a room through the use of the webcam. The user only needs to log on, and then start monitoring. At the launch he has the option to enable streaming, which enables the user via another machine to access the images from his webcam. Upon detection of intrusion, an email is sent to the user with the captured image. The languages used for this application are the JavaScript, especially to encode the server and the PHP and HTML. For coding the server we used the NodeJS framework.

Keywords : NodeJS, JavaScript, HTML5,Security

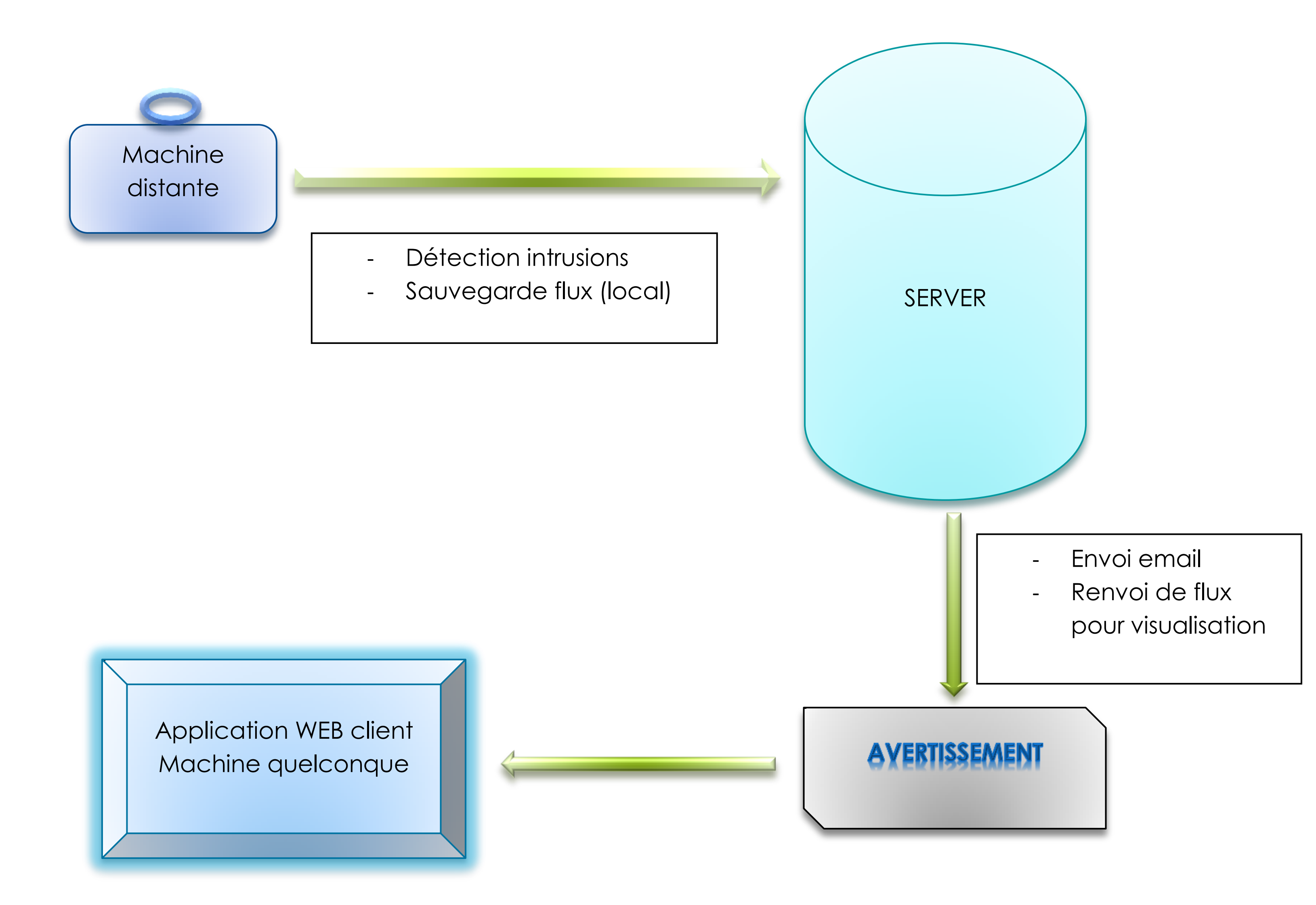
ANNEXES

1. Présomption de faisabilité

a) Cas1 : pas intrusions



b) Cas 2 : Detection d'intrusions



c) Solutions techniques

- ✓ Coté application web :

  - Page web classique avec possibilité de création de compte (utilisateurs et mot de passe associé gérer par une BDD)
  - Connexion à distance sur le site désiré
  - Codage en PHP et type de BDD selon le choix du programmeur pour la gestion des comptes
  - Design du site selon le choix du codeur.
  - (Utilisable que sous Google chrome (pour l'instant))

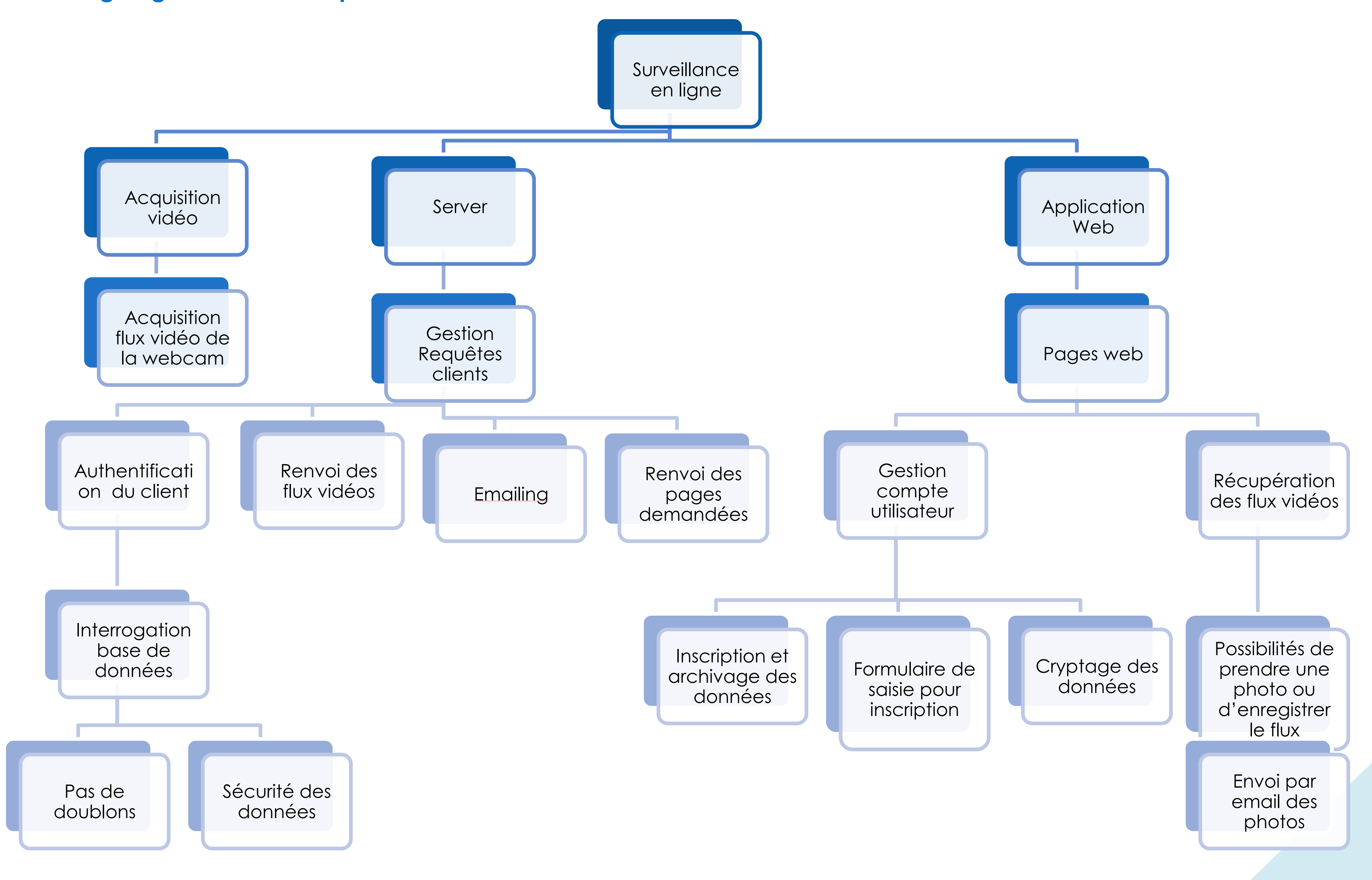
- ✓ Coté serveur :

  - Utilisation de nodejs (possibilité d'utiliser JavaScript coté serveur)
  - Codage du server dans un autre langage (efficacité et utilité à prouver)
  - Lieu d'hébergement
  - Possibilité de sauvegarde des flux
  - Rapide
  - Pas de blocage du nombre de client
  - Réponses aux requêtes instantanées

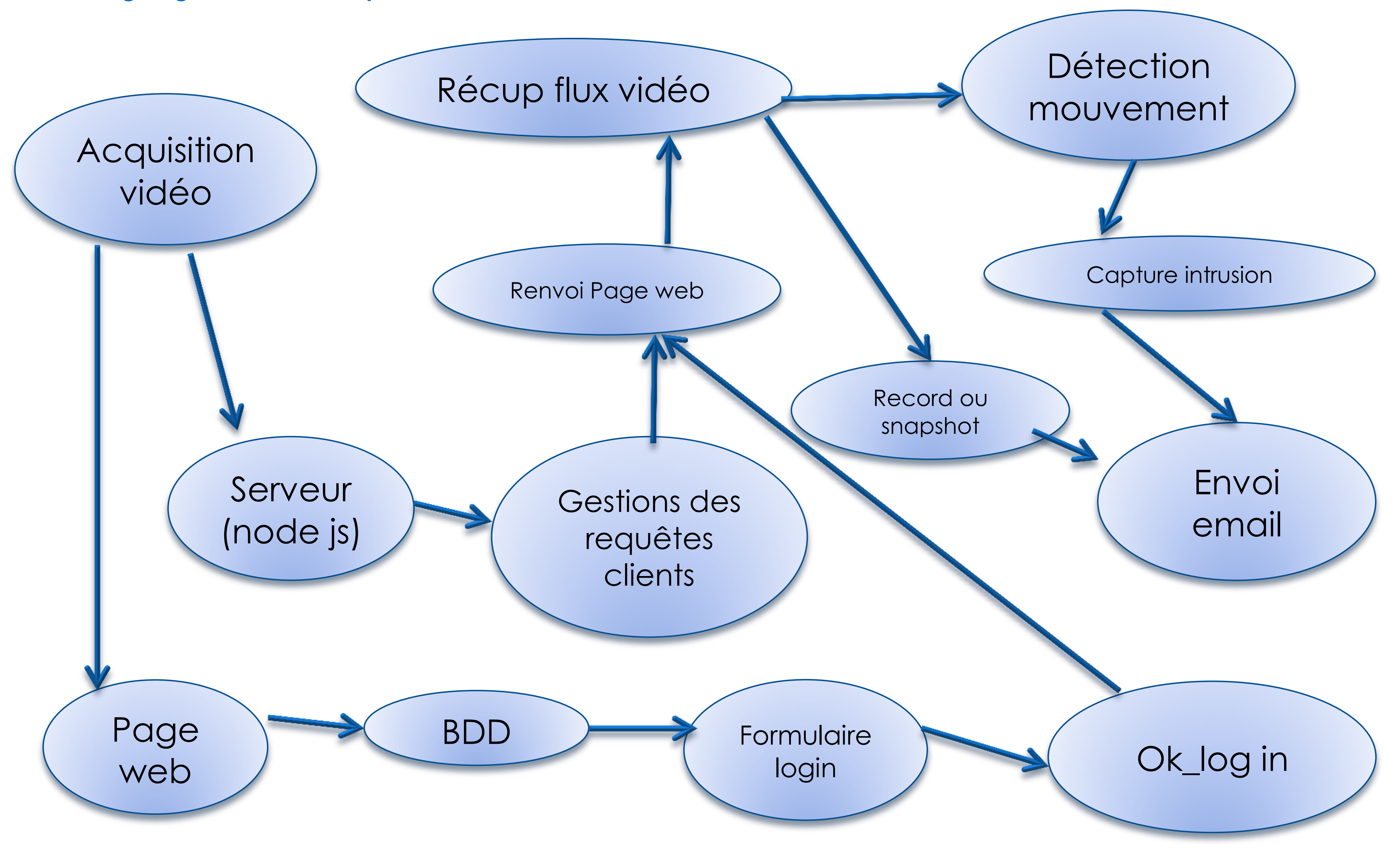
2. METHODE DES ANTECEDENTS

N°	Tâches	Succ.	Durées (heures)	Début au plus tard
1	Acquisition vidéo	2,3	5	Séance 1
2	serveur	3,9,10	20	Séance 3
3	Page web	4	1	Séance 6
4	Gestion des requêtes	5	10	Séance 8
5	Renvoi des pages	4	10	Séance 10
6	Récupérations flux	7,8	12	Séance 10
7	Enregistrement vidéo ou prise de photos	14	6	Séance 12
8	Détection de mouvement	14	8	Séance 8
9	Base de données	10, 11	6	Séance 6
10	Formulaire d'inscription	11	6	Séance 7
11	Archivage des données d'inscription	12, 13,14	2	Séance 8
12	Sécurité des données	aucun	1	Séance 9
13	Authentification client	aucun	8	Séance 9
14	Envoi email	aucun	12	Séance 14

3. Organigramme technique



4. Organigramme technique 2



5. Diagrame PERT

